A Research Synopsis on

"Improvement in methodologies for Detection of Software Vulnerabilities using Deep-Learning Techniques"

For the registration of Ph.D. degree in subject of Computer Science

Submitted By

Mr. Yogesh Vasantrao Hushare

Under the Guidance of

Dr. Sudhir B. Jagtap

Principal,

Swami Vivekanand Mahavidyalaya,
Udgir Dist. Latur

Submitted to

Swami Ramanand Teerth Marathwada University, Nanded

1. Introduction:

Software vulnerability has long been a severe but crucial research issue in cybersecurity [1–3]. These security vulnerabilities threaten the IT infrastructure of organizations and government sectors. There are increasingly more vulnerabilities being discovered. Multiple vulnerabilities released in the Common Vulnerabilities and Exposures were approximately 4,600 in 2010. However, it grows to approximately 153,955 in 2021. Software vulnerability [4–6], as a threat, is increasing in frequency, scale, and severity, which are similar to natural disasters; it may lead to unintended and severe consequences. Once vulnerability in a key system is exploited by attackers, millions of computer systems may be affected [7].

Software vulnerability can cause disastrous consequences for information security. Earlier detection of vulnerabilities minimizes these consequences. Manual detection of vulnerable code is very difficult and very costly in terms of time and budget. Therefore, developers must use automatic vulnerabilities prediction (AVP) tools to minimize costs. Recent works on AVP begin to use techniques of deep learning (DL)**.**

The term **open source** refers to something people can modify and share because its design is publicly accessible. Open-source software is software with source code that anyone can inspect, modify, and enhance. Prime examples of open-source products are the Apache HTTP Server, the e-commerce platform of Commerce, internet browsers Mozilla Firefox and Chromium and the full office suite LibreOffice.

Open-source vulnerabilities are basically security risks in open-source software. These are weak or vulnerable code that allows attackers to conduct malicious attacks or perform unintended actions that are not authorized. In some cases, open-source vulnerabilities can lead to cyberattacks like denial of service (DoS). The WhiteSource Vulnerabilities Database listed the following top 10 vulnerabilities observed in Open-source software's.

1. Lodash
2. FasterXMLjackson-databind
3. HtmlUnit
4. Handlebars
5. http-proxy
6. decompress
7. XStream
8. Netty
9. Spring Framework
10. PyYAML

Deep learning-based vulnerability detection has attracted much attention recently. These techniques have been applied in the field of communications and networking and have achieved promising outcomes. For vulnerability detection tasks, neural networks are applied for automated feature extraction, which helps to improve the generalization ability of a model being capable of extracting high-level and latent features automatically. Therefore, researchers are motivated to improve the usefulness of deep learning-based vulnerability detection solutions from various aspects. For vulnerability detection tasks, neural networks are applied for automated feature extraction, which helps to improve the generalization ability of a model being capable of extracting high-level

and latent features automatically. Therefore, researchers are motivated to improve the usefulness of deep learning-based vulnerability detection solutions from various aspects.

The process of applying deep learning techniques in the context of vulnerability detection can be divided into four steps: data collection, data preparation, model building, and evaluation/test. Some existing vulnerability detectors with traditional embedding solutions such as Word2Vec, GloVe, and FastText often incur low precision and recall [11]. The bimodal feature of code [12] demands a model to be able to handle long-distance contextual dependencies. However, conventional embedding methods adopted by existing studies, such as Word2Vec, can only generate a unique embedding for a given code token and are unable to produce embeddings based on different contexts. Hence, we consider usingCodeBERT as a code embedding and feature generator.

2. Literature Review:

The comparison between proprietary software's and open-source software's leads to understand the need of recent work domains. The open-source approach is certainly a very effectiveand rewarding way to involve people in a softwaredevelopment project. So, study of distinctive features and characteristics need to be done to explore the application and usefulness of the various open-source software's [2].Software can be termed open source independentof how or by whom it has been developed: the termdenotes only the type of license under which it ismade available.Now a days, an open-source software development projectis typically initiated by an individual or a small groupwith an idea for something interesting they themselveswant for an intellectual or personal or business reason [5].

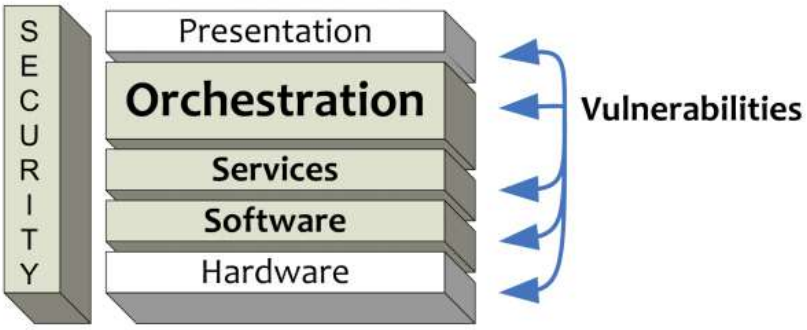
The DeLone and McLeanIS success model isa widely accepted model for evaluating IS success.This model is being efficiently used to measure the success of OSS. By using IS successmodel to the OSS context and is has been observed that the OSS success model shared similarities and differences with othercontexts [6].The purpose of the Free Software Foundation is notto ensure distributing software to the end user withoutcost, but to ensure that the end user can use the software freely.The generalopen-source systemdevelopment cycle allows multiple participants tocontribute to the software developmentprocess requires a massive coordinationeffort[3].Based on the importance of the end user andthe practice of incremental release, OSS is often compared to Rapid Application Development. Certainly, Although RAD approach of software development has its own advantages, OSS approach also proving the importance in scope, availability of source code and budget concern [4].

A lot of research has been done on Software Vulnerabilities, the best way to review the literature to improvise Software Vulnerabilities Detection it is important do the analysis of work done. The study of variousvulnerability detection techniques such as Lexical Analysis, Data Flow Analysis, Abstract Interpretation,Model Checking, Fuzzing, Taint Analysis, Symbolic Execution and hybrid methods will be useful to move ahead in the advance research[1].

It is important to highlight the recent software vulnerabilities at regular basis to avoid the risks, but the disclosing the detected vulnerabilities is also area of concern from the developer's point of view.The market-based infomediary provides monetary rewards to identifiers for each vulnerability reported.When the market-based infomediary leaks vulnerability information, it is referred as the unregulatedmarket-based

mechanism.When voluntary disclosure is low, encouraging a market-based mechanism with some regulation is a good idea.CERTtype infomediary and the market-based infomediary(where they exchange vulnerability information) thatmay lead to better results than the current environment, in which CERT and the market-based firm donot cooperate [7].

An approach of extrapolating SOAvulnerabilities from previous vulnerability collections and
classifying the resulting vulnerabilities according to theirimpact on the business process.A SOA vulnerability is a vulnerability present in such anenvironment. Vulnerabilities can exist in any of the SOAlayers:



Focus lies on the orchestration layer, as it representsthe biggest change a SOA brings. The
orchestration layer contains the business process logic. Vulnerabilities which allow an attacker tomodify either the activities or their sequence in a businessprocess are our primary topic of interest.he typical approach to creating a SOA vulnerabilityclassification would be to wait until a considerable amountof vulnerabilities has been collected, and then look for similarities based on which classes could be defined. Main approach is to examine existing vulnerabilities regarding their "survivability" in a SOA [8].

The vulnerability discovery model(VDM), considered the impact of two factors that governed the rate of change of vulnerability discovered. The first factor deals with the number of installed base due to the rising popularity of a software and the second factor capture the decreasing phenomenon of the number of undetected vulnerabilities with time. First factor constitute the vulnerabilities discovered with a detection rate r and the second factor represents additional vulnerabilities which are detected due to the influence ofthe vulnerabilities discovered by time t. The differential equation describing the discovery process can be modeled as

$$\frac{d\Omega(t)}{dt} = r\left(N - \Omega(t)\right) + s\frac{\Omega(t)}{N}\left(N - \Omega(t)\right)$$

The predictability of the proposed model[9]has been assessed by fitting the VDM to an observed sample and evaluate the goodness-of-fit criterion of the fitted model on the observed samples to predict the future behaviour of the vulnerabilities. Non-linear least square methodology has been used to evaluate the estimation procedure on the security vulnerability data set. The results show a better insight about the vulnerability discovery process and revealed that it is better to use the proposed s-shaped model to estimate the

vulnerabilities.

The new features of Deep Learning (DL), especially indomains like computer vision and natural language processing, has sparked interest in using DL to detect securityvulnerabilities automatically with high accuracy. Deep Learning-based Vulnerability Prediction (DLVP) techniques have been reported to detect security vulnerabilitieswith high accuracy DLVP methods aim to detect unknown vulnerabilities intarget software by learning different vulnerability patternsfrom a training dataset. Although Deep Learning is playing vital role in vulnerabilities detection then also Key limitations of existing DLVP approaches that observed are: Data Duplication, Data Imbalance, Learning Irrelevant Features, Model Selection: Lack of Class Separation, etc. [10].

Applying deep learning for vulnerability detection is a developing field now a day. To better understand the trends in this field, we firstly identify these game changers with highlights of concepts, ideas, and deep learning models. Because of their unique and significant contributions, we clearly articulate how the subordinate works further explore the field. First proposed deep learning algorithm for learning the semantics code representations indicative of defective code. End-to-End Prediction of Buffer Overruns from Raw Source Code via Neural Memory Networks is used for vulnerability detection. VulDeePecker is A Deep Learning Based System for Vulnerability Detection is one of the beneficial methods. a deep learning-based vulnerability detector for binary code. It aims to broaden the application field of vulnerability detection by mitigating the unavailability of source code. It is neural model-based vulnerability detection from source code to binary code [11].

The Bidirectional Long Short-Term Memory model outperforms LSTM model by a little margin. The decay kinematics of top-quark pairs created in high-energy proton-proton collisions are presented as a probabilistic reconstruction utilizing machine learning. Loss rate of BLSTM is less, accuracy level is more. The sequences of C library function calls as thedataset to create neural network models to forecast vulnerabilities. VulDeePecker trains neural networks to discover vulnerabilities using code gadgets, which are computer statements that are data or control reliant on one other. In the field of cybersecurity, software vulnerability has long been an important but critical research topic. Machine learning (ML)-based approaches have recently sparked increased interest in software vulnerability detection research [12].

Vulnerability prediction models (VPMs) used software metrics as input to discriminate between vulnerable and clean source entities. The main limitation of the VPMs built using software metrics is that they do not have the capacity of pinpointing the exact location of vulnerabilities because metrics are calculated at a coarse level of granularity. Therefore, researchers begin to investigate automatic vulnerability prediction (AVP) using techniques of deep learning. Techniques used in automatic defect prediction are also used in automatic vulnerability prediction (AVP). One of these techniques is Defect Prediction Models (DPM). The proposed VPMs allow vulnerability prediction at a coarse granularity: package, file, class, and function. This can lead to discovering the exact location of vulnerable code [13].

Static analysis, dynamic analysis and Manual code review are the conventional vulnerability detection methods for Open-source software. The issues with the

conventional methods in the main are that static analysis produces too many false positives, dynamic analysis does not scale, and code reviews are time consuming. Emerging detection methods such as Distributed demand-driven security testing, Use of Execution Complexity Metric, IDE Plugins for Early Detection, Machine Learning and Further Knowledge Formalisation and Linking Repositories are used. machine learning, early detection IDE plug-ins and linking repositories show much promise for future work. Machine learning lends itself well to feature rich OSS which speeds up classification of vulnerable code and reduces the time burden on development teams. Early detection IDE plug-ins will help developers implementing OSS to grow and consolidate their secure coding knowledge [14].

The presence and key themes of security issues in GitHub repositories. a posted issue on GitHub that contains a security-related aspect. A mixed-methods approach is used. Collected the issues from 200 randomly sampled GitHub repositories and used them as data corpus. Mansooreh Zahedi et al. [15] mentioned GitHub as the main source of data. Using a crawling application, issues are collected from 200 randomly selected GitHub repositories. All the repositories had at least 500 commits and at least 100 pull requests or issues. This strategy was taken to ensure that the data could provide an overview of the prevalence of security issues on GitHub. Topic modelling is a Natural Language Processing (NLP) technique to automatically extract topics out of a corpus of textual data. The LDA algorithm on the pre-processed data using the "LDA()" function from the "topic models" package. Qualitatively analysed a sample of security issues.

An improved feature selection method is proposed to reveal optimal feature sets by enhancing the classification accuracy of static analysis. Proposed method is the immune-based feature selection that is utilized to discover optimal feature sets by improving the classification accuracy of static analysis. Experiments are conducted on the three standard datasets. The proposed feature selection method is presented and tested to determine the benefit rate of choosing relevant features and to study the influence on the classification of vulnerable components. Static analysis (SA) tools are simple to style checkers or follow intricate rules to find developers' issues often overlook efficiently [16]. Immune system is adaptive and self-organizing by its nature. The clonal selection classification algorithm (CSCA) replacesclonal selection algorithm to improve the algorithm's overall search accuracy and efficiency to improve the update efficiency of the antibody collection. The Immunos-1 algorithm represents an artificial immune system-based algorithm. The artificial immune recognition system (AIRS) is among this kind of supervised learning AIS that has succeeded considerably in a wide variety of classification problems. vulnerability prediction modelling (VPM) techniques and leverage predictive modelling techniques as vulnerable code components.

The design and implementation of a deep learning-based vulnerability detection system, called
Vulnerability Deep Pecker (VulDeePecker). Deep learning has a great potential because deep learning does not need human experts to manually define features, meaning that vulnerability detection can be automated. VulDeePecker is able to detect 4 vulnerabilities, which are not reported in the National Vulnerability Database (NVD). A code gadget is composed of a number of program statements, which are semantically related to each other in terms of data dependency or control dependency. VulDeePecker has two phases a learning (i.e., training) phase and a detection phase. VulDeePecker is limited to dealing with vulnerability detection by assuming source code of programs is available. It only

deals with vulnerabilities related to library/API function calls. It is limited to the BLSTM neural network [17].

A dataset that correlates diverse software metrics derived from thousands open-source components with their (known) security bugs. the construction process of a dataset that correlates software metrics derived from 8694 open-source components with their security vulnerabilities, the dataset and, how it can be used to produce research results. First collected
and processed the various vulnerability reports from the National Vulnerability Database, then cloned the repositories of the selected projects. With the repositories at hand, we mapped version references. Based on the mapping, we checked-out specific versions from the repository. CVE(Common Vulnerabilities and Exposures) contains vulnerabilities, their description, and metrics that involve their severity. CWE stores the vulnerability classification list which consists of the 716 categories and their corresponding descriptions. Project Releases entries represent a project version. Vulnerable Cases contains mappings between vulnerabilities and specific project versions. Code Metrics contains the metrics that we collected for every mapped version of a project during the analysis step. The dataset, together with the scripts used for its construction will be made publicly available on GitHub [18].

Method of finding potentially dangerous code in code repositories with a significantly lower false-positive rate than comparable systems. Conducted the first large-scale mapping of CVEs to GitHub commits in order to create a vulnerable commit database. Based on this database, VCCFinder, a code analysis tool that flag suspicious commits by using SVM-based detection model. This method outperforms Flaw finder by a great margin, reducing the false positives by over 99 % at the same level of recall. Henning Perl et al. [19] construct the first large-scale database mapping CVEs to vulnerability-contributing commits (VCCs). No large-scale database exists those maps vulnerabilities as reported by CVEs to commits. To create a freely available database for ourselves and the research community, a method to automatically map CVEs to vulnerability-contributing commits is created. Then developed and tested a heuristic to proceed from these fixing commits to the vulnerability-contributing commits (VCCs). It is found only three cases (i.e., 3.1 %) where our heuristic blamed a wrong commit for the vulnerability. All three of the mis-mappings occurred in very large commits.

3. ResearchOutline

This section will examine the research question, aim, objectives, scope, significance, limitations and the justifications for this study.

3.1 ResearchQuestion

The research questions examined by this thesis are:

- Given the widespread use of open-source software in today's world, how important is software vulnerability detection?
- How to make use of Deep Learning science and various techniques to detect the vulnerabilities in open-source software’s more precisely and accurately?

3.2 Aim

The aim of this research is to develop a technique using Depp Learning methodology to detect the vulnerabilities present in open-source software’s very precisely. By making use of this research one can avoid the security risks of the most frequently used open source software’s and developers also take this as a tool while developing the software’s.

3.3 Objectives

1. Study of Open-source software Development and analysing the importance of OSS in the current era of Information Technology in different application levels.
2. Study of Vulnerabilities that are detected in OSS and how the graph is increasing day by day when counting the newer vulnerabilities detected.
3. Give exposures to recent vulnerabilities in Open-source software.
4. Do the comparative study of Deep Learning technologies to detect software vulnerabilities.
5. Get the empirical knowledge of Deep Learning technologies to detect vulnerabilities especially in Open-source software.
6. Find improvised Deep Learning implementation to meet the high-performance vulnerabilities detection in Open-source software, so to promote the use of OSS widely.

4. Methodology:

The research will adopt a methodology as the main focus is to achieve its objectives. Firstly, detailed review of existing literature of Software Vulnerabilities will be performed to identify varieties of flows in open-source software'sthose are get addressed and not addressed in the various studies. Method will be evaluated by doing analysis of software vulnerability data and their risks. By making use of various tools available this data will be get analyzed and recent software vulnerabilities will be highlighted. Deep Learning techniques will be implemented in order to improve the software vulnerabilities in open-source software. Conclusions will be drawn on the basis of the above experimentation of implementing Deep Learning techniques and improvised technique will be proposed. Lastly discussion will be done leading to final conclusions and scope for future work.

5. Plan ofwork

5.1 Work Plan

Phase 1 (5-10 months):

Literature survey of Software Vulnerabilities, Open-source Software's, Software Vulnerability Detection Technique, Algorithms of Deep Learning, etc.

Phase 2 (2-5 months):

Detail study of Tools and Techniques used to detect software vulnerabilities and getting more knowledge to make use of Deep Learning for open-source softwarevulnerabilities Detection.

Phase 3 (10-15 months):

Investigationdetected software vulnerabilities and the their impact on specific open source software's. Proposing improvements / new algorithm to do the Software Vulnerabilities Detection.

Phase 4 (15-20 months):

Validation of improvement/ new algorithm proposed to do the Software Vulnerabilities Detection.

Phase 5 (5-10 months):

Analyze the result obtained and thesis writing.

5.2 Ganttchart:

Project Plan	Months											
	5	10	15	20	25	30	35	40	45	50	55	60
Phase 1												
Phase 2												
Phase 3												
Phase 4												
Phase 5												

6. References:

1. YunfeiSu,MengjunLi,Chaojing Tang, Rongjun Shen, An Overview of Software Vulnerability Detection,International Journal of Computer Science and Technology,Vol. 7, Issue3, July-Sept 2016.
2. Alfonso Fuggetta, Open source software—an evaluation, The Journal of Systems and Software 66 (2003) 77–90.
3. D Ming-Wei Wu,Ying-Dar Lin, Open-Source SoftwareDevelopment: An Overview, IEEE, Jun 2001.
4. Feller, Joseph; Fitzgerald, Brian, A framework analysis of the open-source software development paradigm, Proceedings of the twenty first international conference on Information systems, Brisbane,Queensland, Australia, 10-13 December., pp. 58-69.
5. Georg von Krogh,Eric von Hippel, Special issue on open-sourcesoftware development, Research Policy 32 (2003) 1149–1157.
6. Sang-Yong Tom Lee, Hee-Woong Kim, Sumeet Gupta, Measuring open-source software success, Omega 37 (2009) 426 – 438.
7. Karthik Kannan, Rahul Telang, Market for Software Vulnerabilities? Think Again, Management Science, Vol. 51, May 2005, pp. 726-740.
8. Lutz Lowis and Rafael Accorsi,On a Classification Approach for SOA Vulnerabilities
9. Navneet Bhatt, Adarsh Anand, V. S. S. Yadavalli, Vijay Kumar, Modeling and Characterizing Software Vulnerabilities, International Journal of Mathematical, Engineering and Management Sciences, Vol. 2, No. 4, 288–299, 2017.
10. Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, BaishakhiRay,Deep Learning based Vulnerability Detection:Are We There Yet?,IEEE Transactions On Software Engineering, VOL. TBD, 2020.

11. Peng Zeng, Guanjun Lin, Lei Pan, Yonghang Tai, And Jun Zhang, Software Vulnerability Analysis and DiscoveryUsing Deep Learning Techniques: A Survey, IEEE Access, Vol. 8, 2020.

12. Gourav Bansal, Kurukshetra University, Deep Learning to Detect Software Vulnerabilities, International Journal of Engineering Research & Technology (IJERT), Vol. 11 Issue 02, February-2022.

13. Mohammed Zagane, Mustapha Kamel Abdi, And Mamdouh Alenezi, Deep Learning for Software Vulnerabilities, Detection Using Code Metrics, IEEE Access Vol.8 2020.

14. Stuart Millar,Vulnerability Detection in Open-Source Software:An Introduction

15. Mansooreh Zahedi, Muhammad Ali Babar, Christoph Treude, An Empirical Study of Security Issues Posted in Open Source Projects, Proceedings of the 51st Hawaii International Conference on System Sciences, 2018.

16. Canan Batur Şahin, Laith Abualigah, A novel deep learning-based feature selection model for improvingthe static analysis of vulnerability detection, Article in Neural Computing and Applications· October 2021.

17. Zhen Li, Deqing Zou, Shouhuai Xu, XinyuOu, Hai Jin, Sujuan Wang, Zhijun Deng, Yuyi Zhong, VulDeePecker: A Deep Learning-Based System forVulnerability Detection.

18. AntoniosGkortzis, Dimitris Mitropoulos and DiomidisSpinellis, VulinOSS: A Dataset of Security Vulnerabilitiesin Open-source Systems, MSR'18, May 2018, Gothenburg, Sweden.

19. Matthew Smith, Daniel Arp, Fabian Y amaguchi, Konrad Rieck, VCCFinder, Conference Paper· October 2015, DOI: 10.1145/2810103.2813604.

Mr. Y.V. Hushare
(Research Student)

Dr. Sudhir B. Jagtap
(Research Guide)